

Systeme II

5. Die Transportschicht

Christian Schindelhauer

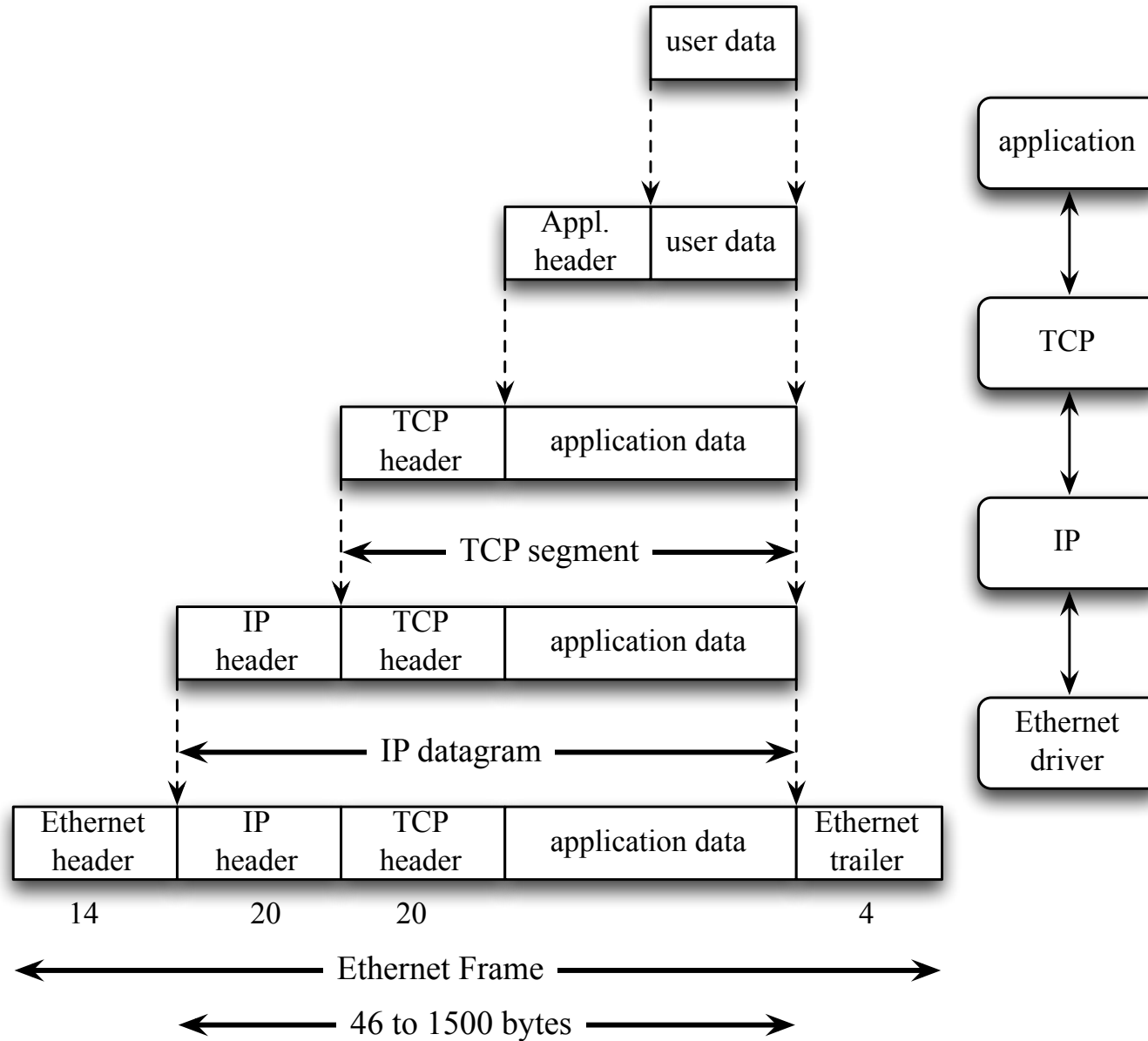
Technische Fakultät

Rechnernetze und Telematik

Albert-Ludwigs-Universität Freiburg

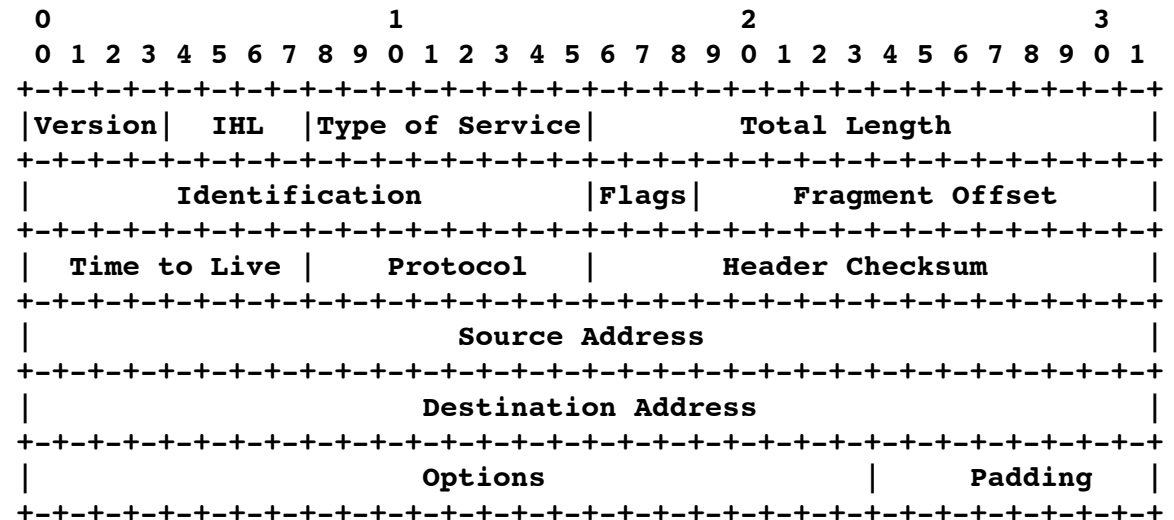
Version 30.06.2014

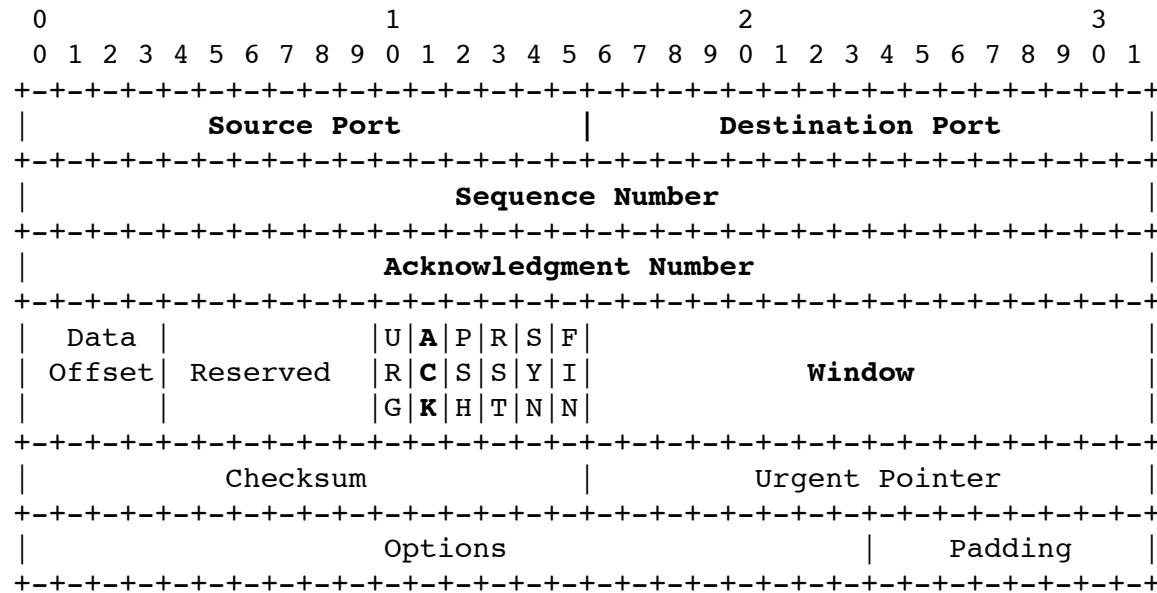
Datenkapselung



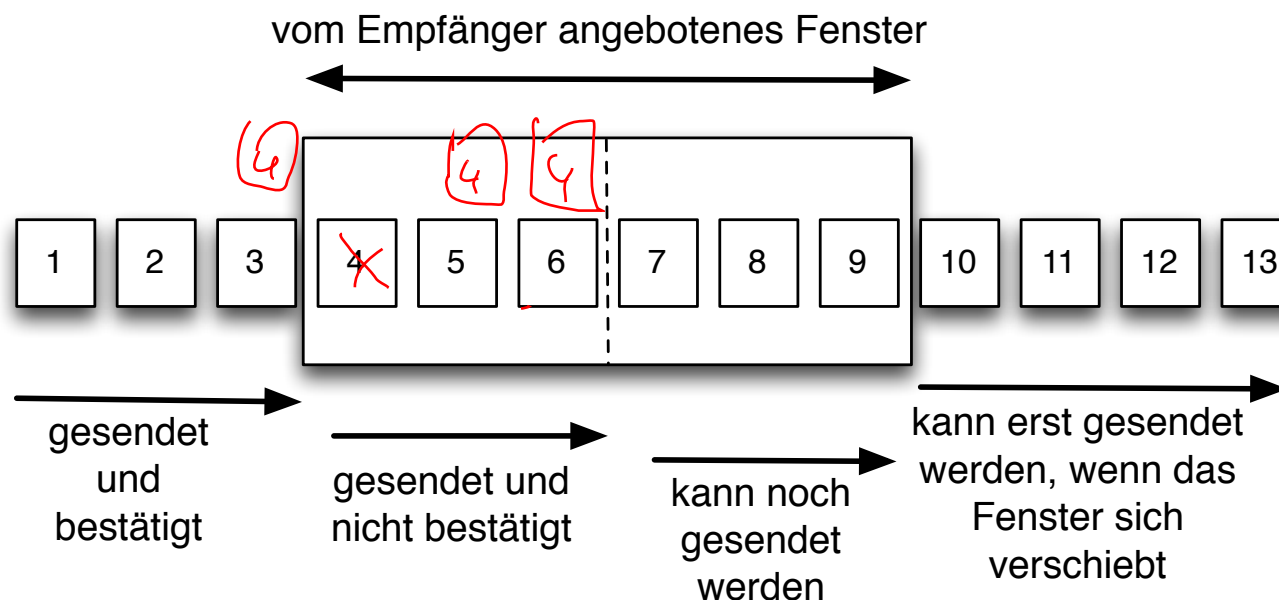
IP-Header (RFC 791)

- Version: 4 = IPv4
- IHL: Headerlänge
 - in 32 Bit-Wörter (>5)
- Type of Service
 - Optimierte delay, throughput, reliability, monetary cost
- Checksum (nur für IP-Header)
- Source and destination IP-address
- Protocol, identifiziert passendes Protokoll
 - Z.B. TCP, UDP, ICMP, IGMP
- Time to Live:
 - maximale Anzahl Hops



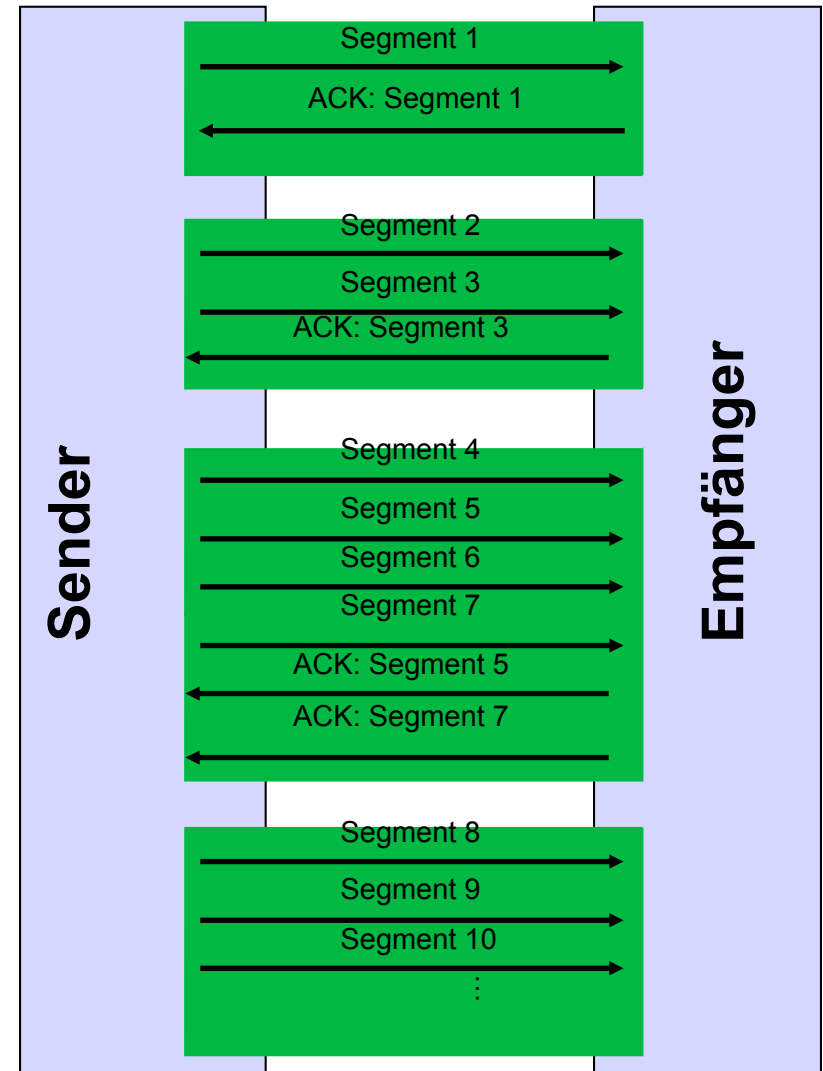


- Datenratenanpassung durch Fenster
 - Empfänger bestimmt Fenstergröße (wnd) im TCP-Header der ACK-Segmente
 - Ist Empfangspuffer des Empfängers voll, sendet er $wnd=0$
 - Andernfalls sendet Empfänger $wnd>0$
- Sender beachtet:
 - Anzahl unbestätigter gesender Daten $\leq$ Fenstergröße



Slow Start Congestion Fenster

- Sender darf vom Empfänger angebotene Fenstergröße nicht von Anfang wahrnehmen
- 2. Fenster: Congestion-Fenster (cwnd/Congestion window)
 - Von Sender gewählt (FSK)
 - Sendefenster: $\min \{w_{nd}, c_{wnd}\}$
 - S: Segmentgröße
 - Am Anfang:
 - $c_{wnd} \leftarrow S$
 - Für jede empfangene Bestätigung:
 - $c_{wnd} \leftarrow c_{wnd} + S$
 - Solange bis einmal Bestätigung ausbleibt
- „Slow Start“ = Exponentielles Wachstum



Stauvermeidung in TCP Tahoe

- Jacobson 88:

x: Anzahl Pakete pro RTT

- Parameter: cwnd und Slow-Start-Schwellwert (ssthresh=slow start threshold)
- S = Datensegmentgröße = maximale Segmentgröße

- Verbindungsaufbau:

- cwnd $\leftarrow$ S ssthresh $\leftarrow$ 65535

x $\leftarrow$ 1

y $\leftarrow$ max

- Bei Paketverlust, d.h. Bestätigungsdauer > RTO,

- multiplicatively decreasing

$$\text{cwnd} \leftarrow S \quad \text{ssthresh} \leftarrow \max \left\{ 2 \times S, \frac{\min \{ \text{cwnd}, \text{wnd} \}}{2} \right\}$$

x $\leftarrow$ 1

y $\leftarrow$ x/2

- Werden Segmente bestätigt und cwnd $\leq$ ssthresh, dann

- slow start: cwnd $\leftarrow$ cwnd + S

x $\leftarrow$ 2·x, bis x = y

- Werden Segmente bestätigt und cwnd > ssthresh, dann additively increasing

$$\text{cwnd} \leftarrow \text{cwnd} + S \cdot \frac{S}{\text{cwnd}}$$

x $\leftarrow$ x + 1

*1S
RTT*

TCP Tahoe

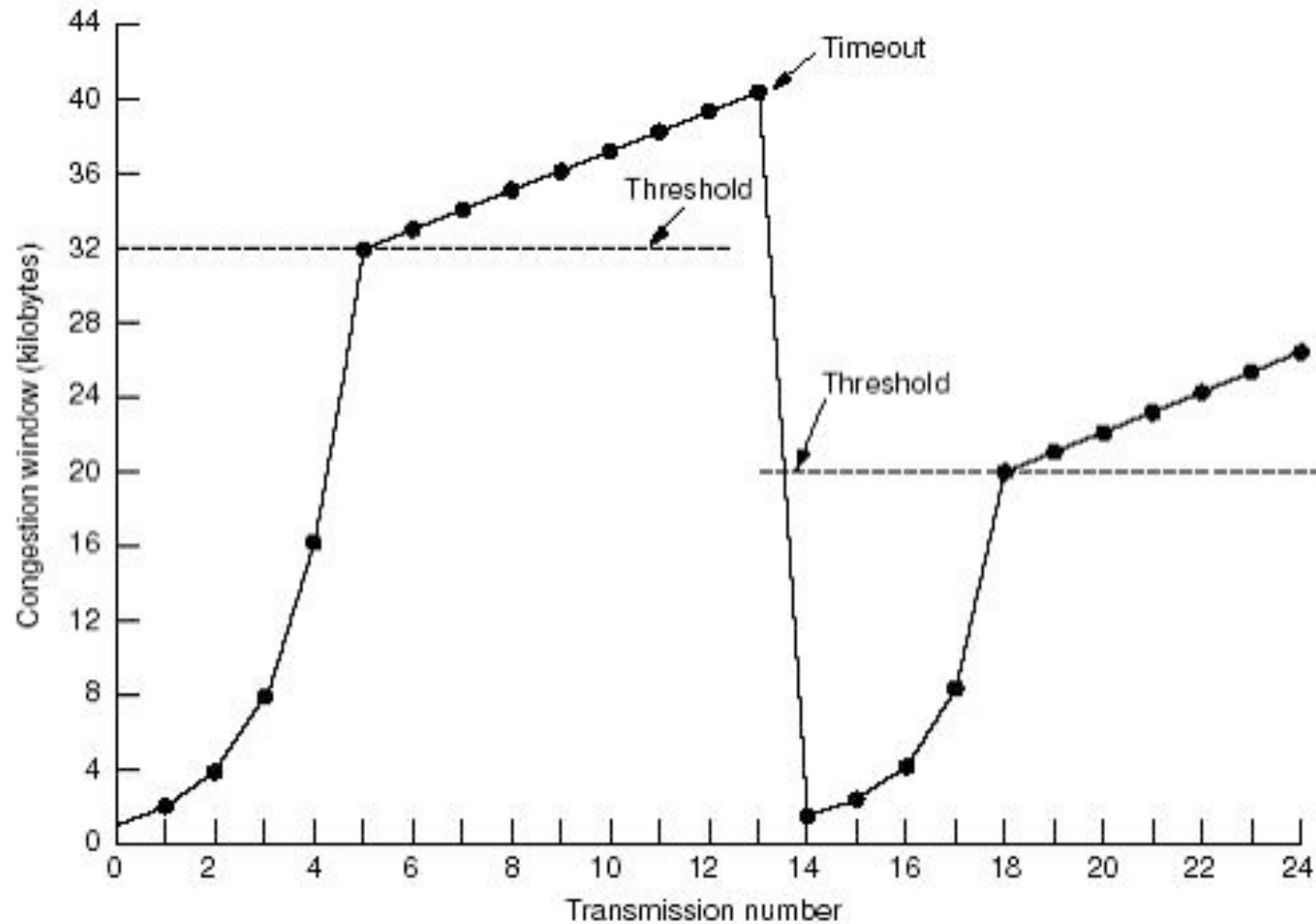


Fig3

pictures from TANENBAUM A. S. *Computer Networks 3rd edition*

- TCP Tahoe [Jacobson 1988]:
 - Geht nur ein Paket verloren, dann
 - Wiederversand Paket + Restfenster
 - Und gleichzeitig Slow Start
 - Fast retransmit
 - Nach drei Bestätigungen desselben Pakets (triple duplicate ACK),
 - sende Paket nochmal, starte mit Slow Start
- TCP Reno [Stevens 1994]
 - Nach Fast retransmit:
 - $\text{ssthresh} \leftarrow \min(\text{wnd}, \text{cwnd})/2$
 - $\text{cwnd} \leftarrow \text{ssthresh} + 3 S$
 - Fast recovery nach Fast retransmit
 - Erhöhe Paketrate mit jeder weiteren Bestätigung
 - $\text{cwnd} \leftarrow \text{cwnd} + S$
 - Congestion avoidance: Trifft Bestätigung von $P+x$ ein:
 - $\text{cwnd} \leftarrow \text{ssthresh}$

$y \leftarrow x/2$
$x \leftarrow y + 3$

Stauvermeidungsprinzip: AIMD

- Kombination von TCP und Fast Recovery verhält sich im wesentlichen wie folgt:

- Verbindungsaufbau:

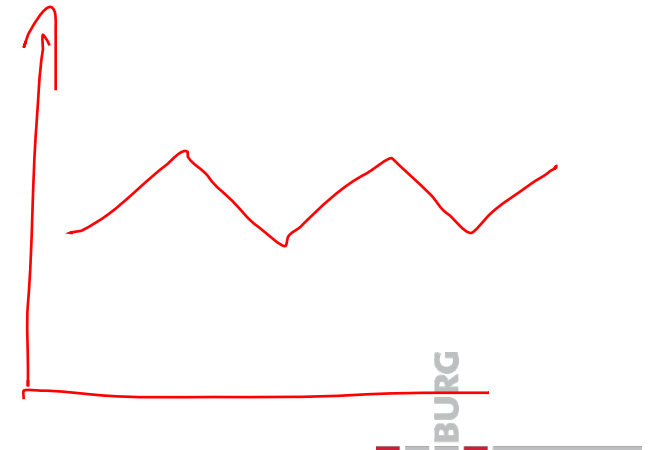
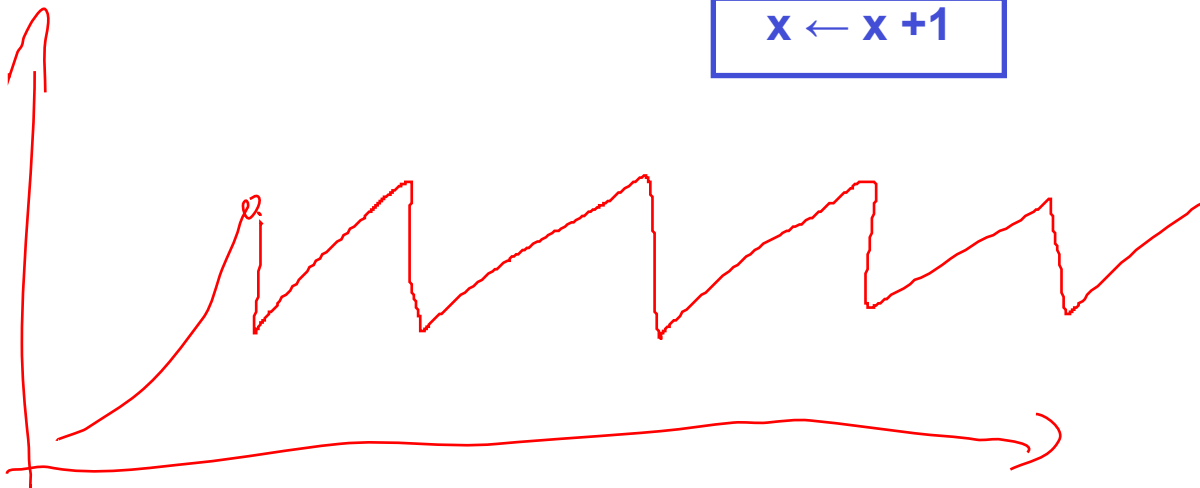
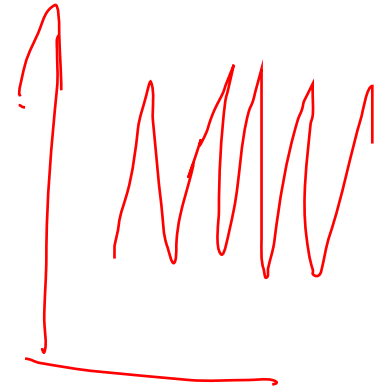
$$x \leftarrow 1$$

- Bei Paketverlust, MD: multiplicative decreasing

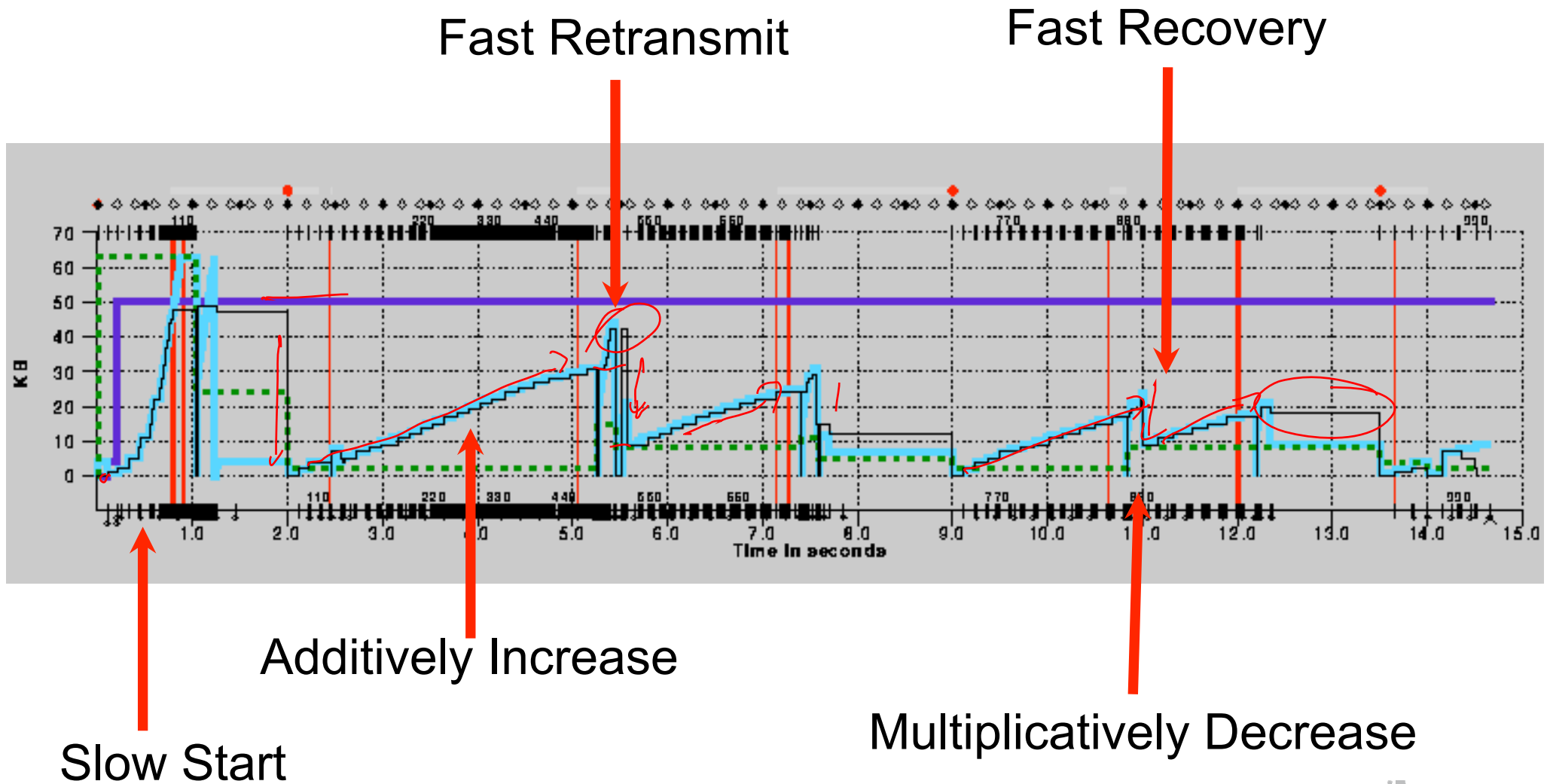
$$x \leftarrow x/2$$

- Werden Segmente bestätigt, ~~AD~~^{AI}: additive increasing

$$x \leftarrow x + 1$$

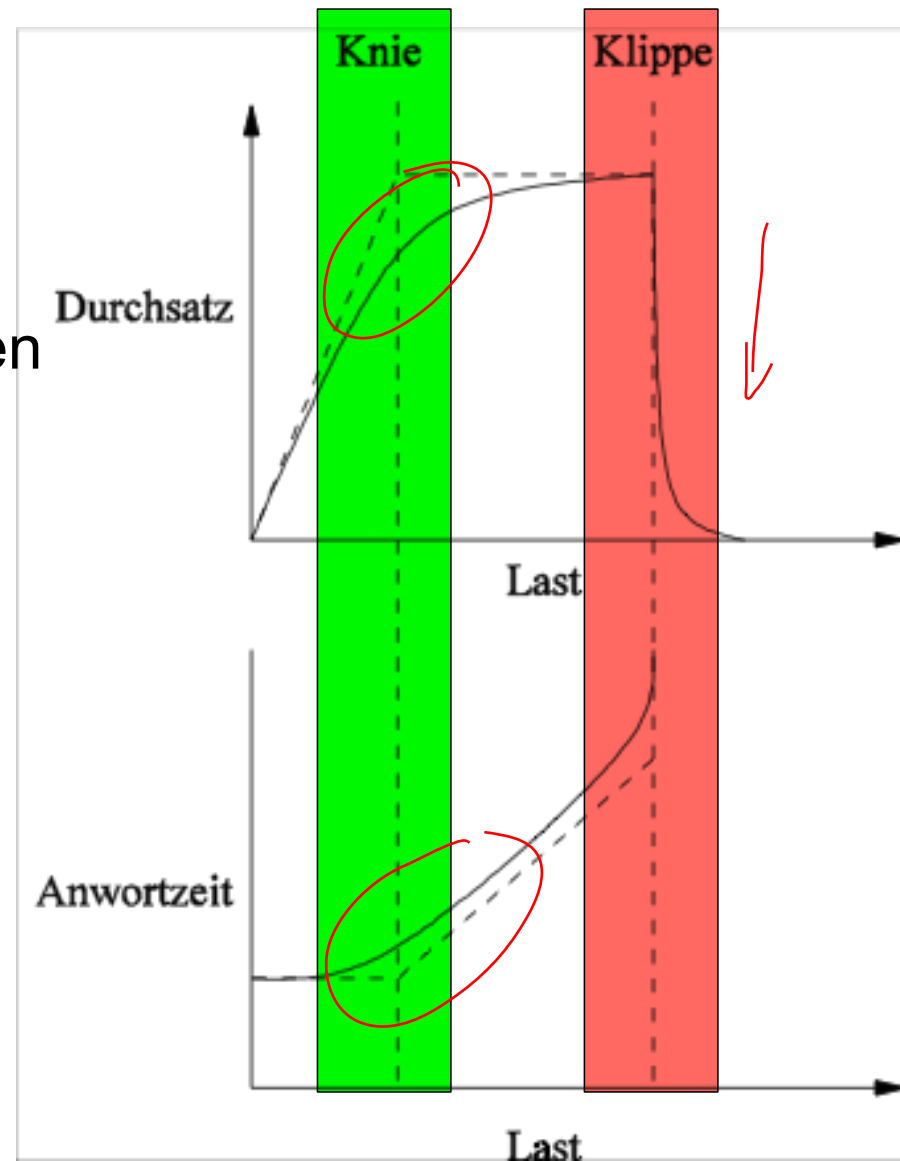


Beispiel: TCP Reno in Aktion



Durchsatz und Antwortzeit

- **Klippe:**
 - Hohe Last
 - Geringer Durchsatz
 - Praktisch alle Daten gehen verloren
- **Knie:**
 - Hohe Last
 - Hoher Durchsatz
 - Einzelne Daten gehen verloren



Ein einfaches Datenratenmodell

- n Teilnehmer, Rundenmodell
 - Teilnehmer i hat Datenrate $x_i(t)$
 - Anfangsdatenrate $x_1(0), \dots, x_n(0)$ gegeben
- Feedback nach Runde t:
 - $y(t) = 0$, falls $\sum_{i=1}^n x_i(t) \leq K$
 - $y(t) = 1$, falls $\sum_{i=1}^n x_i(t) > K$
 - wobei K ist Knielast
- Jeder Teilnehmer aktualisiert in Runde t+1:
 - $x_i(t+1) = f(x_i(t), y(t))$
 - Increase-Strategie $f_0(x) = f(x, 0)$
 - Decrease-Strategie $f_1(x) = f(x, 1)$
- Wir betrachten lineare Funktionen:

$$f_0(x) = a_I + b_I x \quad \text{und} \quad f_1(x) = a_D + b_D x .$$

■ Interessante Spezialfälle:

- AIAD: Additive Increase
Additive Decrease

$$f_0(x) = a_I + x \quad \text{und} \quad f_1(x) = a_D + x ,$$

wobei $a_I > 0$ und $a_D < 0$.

- MIMD: Multiplicative
Increase/Multiplicative
Decrease

$$f_0(x) = b_I x \quad \text{und} \quad f_1(x) = b_D x ,$$

wobei $b_I > 1$ und $b_D < 1$.

- AIMD: Additive Increase
Multiplicative Decrease

$$f_0(x) = a_I + x \quad \text{und} \quad f_1(x) = b_D x ,$$

wobei $a_I > 0$ und $b_D < 1$.

■ Effizienz

- Last:

$$X(t) := \sum_{i=1}^n x_i(t)$$

- Maß

$$|X(t) - K|$$

■ Fairness: Für $x=(x_1, \dots, x_n)$:

$$F(x) = \frac{(\sum_{i=1}^n x_i)^2}{n \sum_{i=1}^n (x_i)^2} \cdot$$

- $\frac{1}{n} \leq F(x) \leq 1$

- $F(x) = 1 \leftrightarrow$ absolute Fairness

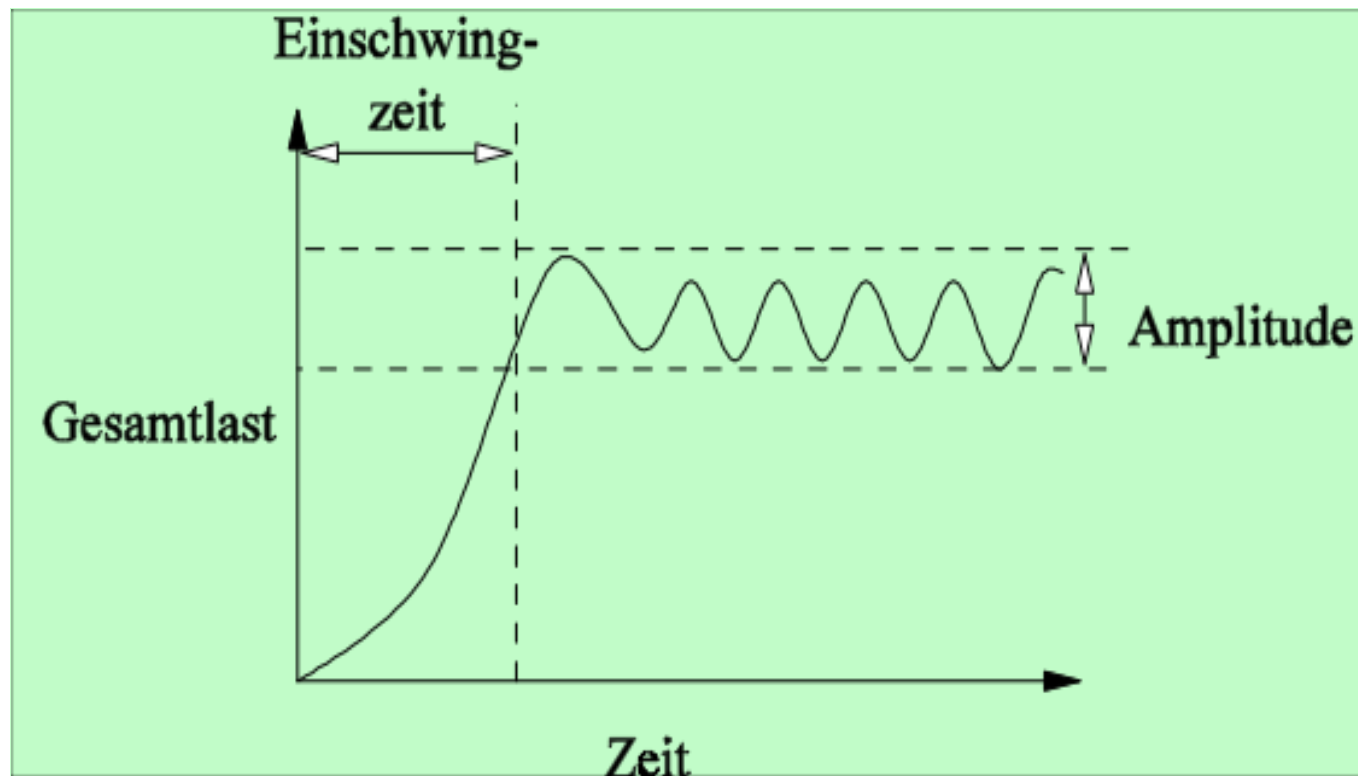
- Skalierungsunabhängig

- Kontinuierlich, stetig, differenzierbar

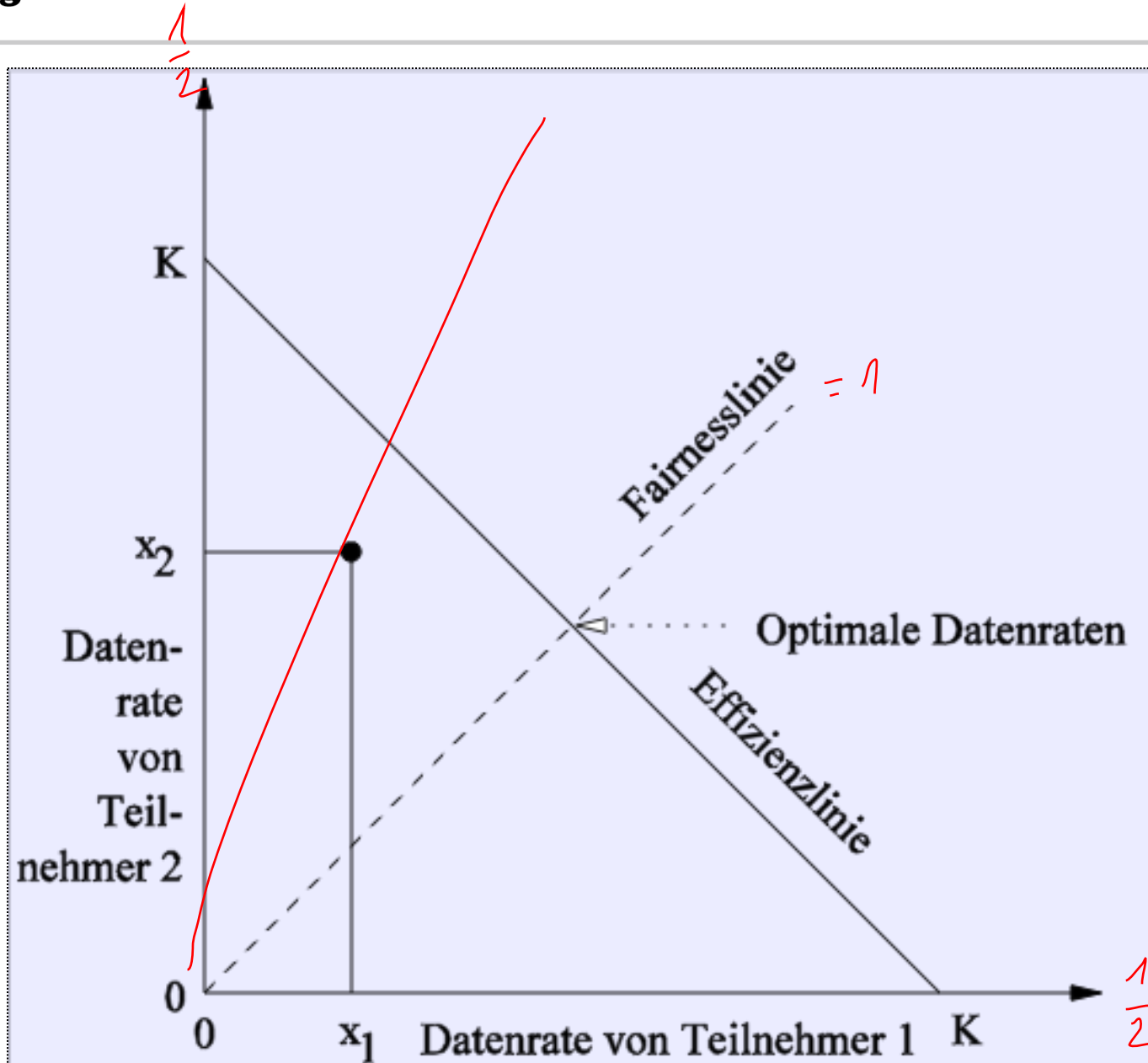
- Falls k von n fair, Rest 0, dann $F(x) = k/n$

$$\frac{\left(\sum_{i=1}^n x_i + 1\right)^2}{n \sum_{i=1}^n (x_i + 1)^2}$$

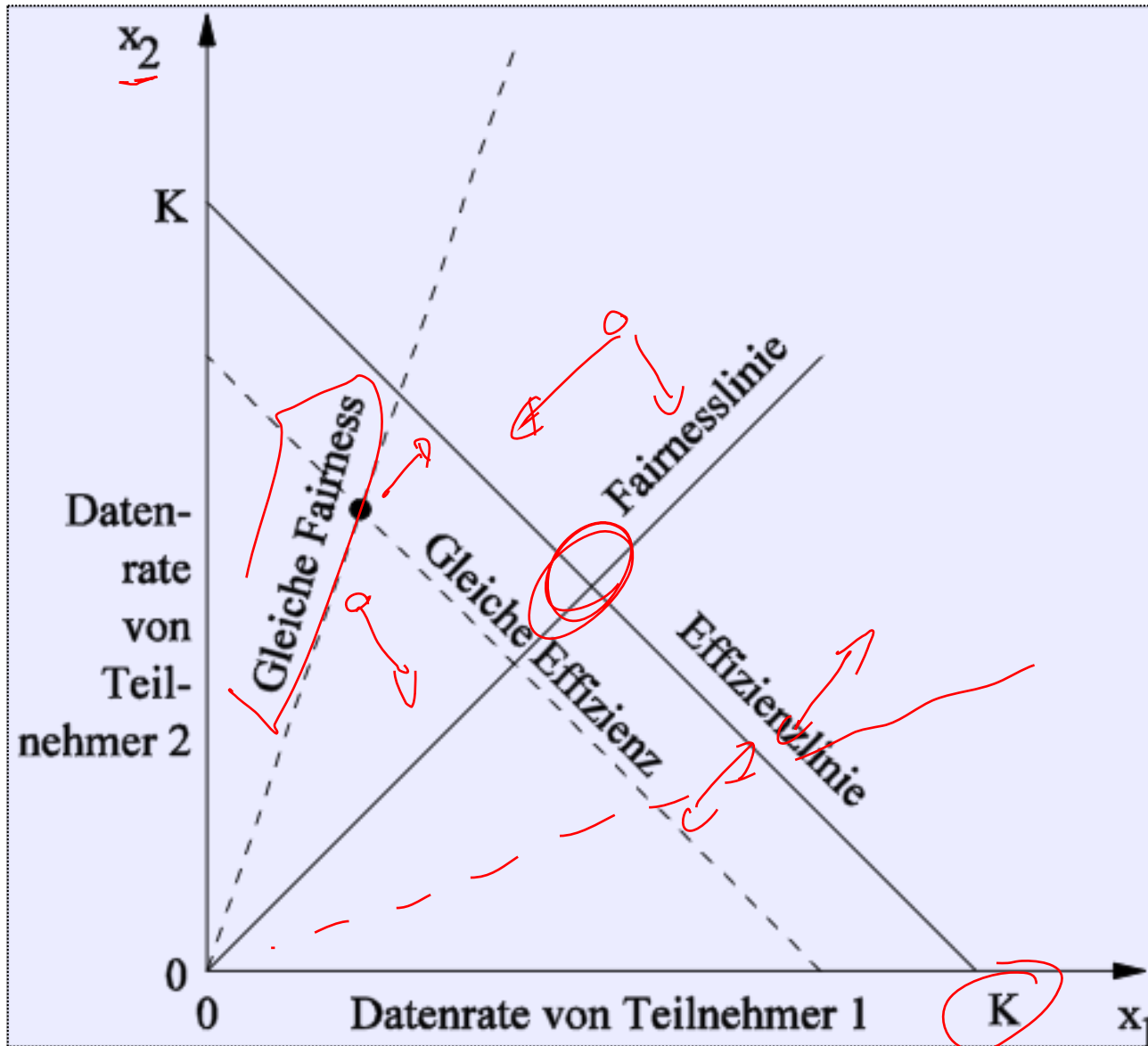
- Konvergenz unmöglich
- Bestenfalls Oszillation um Optimalwert
 - Oszillationsamplitude A
 - Einschwingzeit T



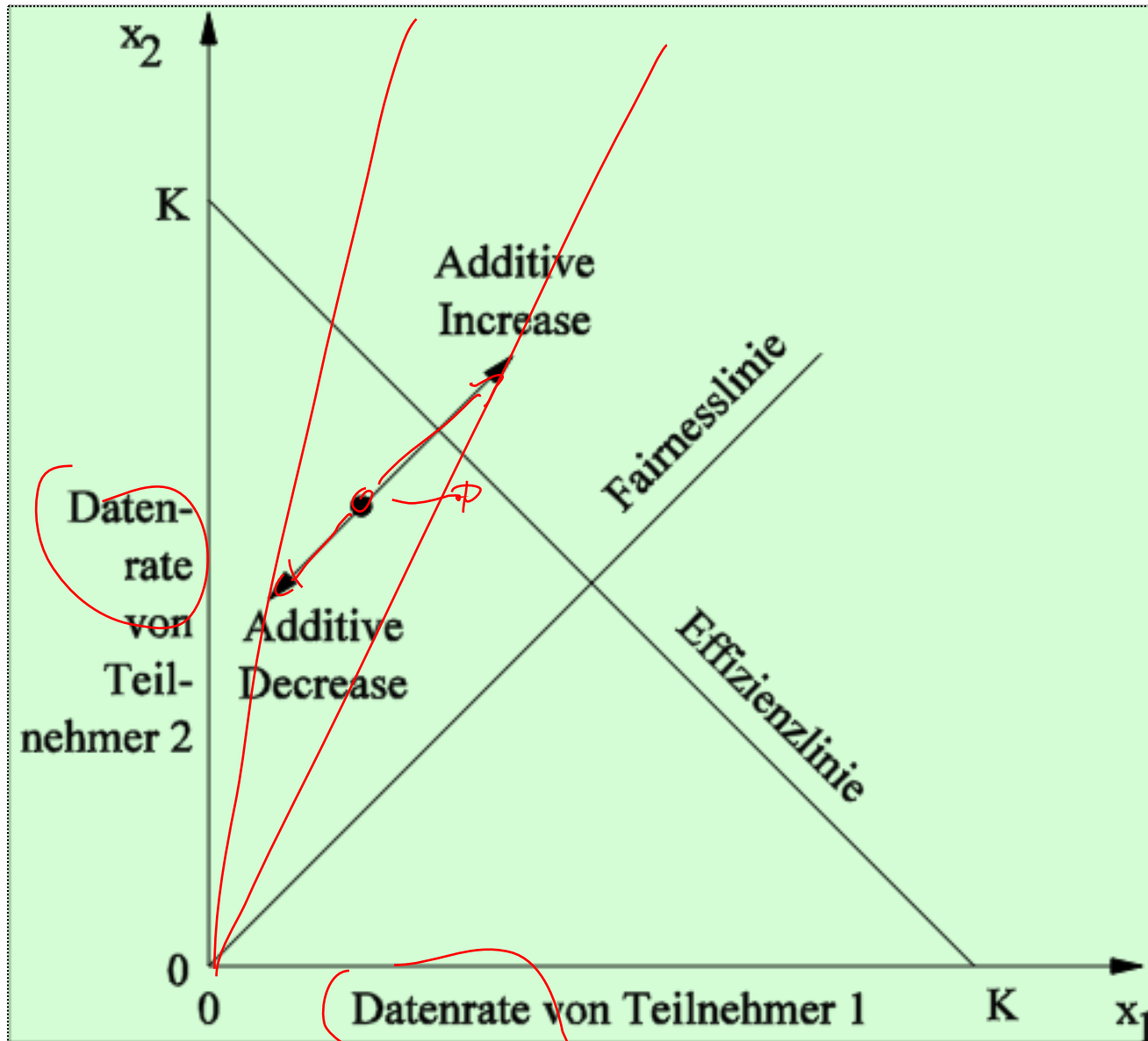
Vektordarstellung (I)



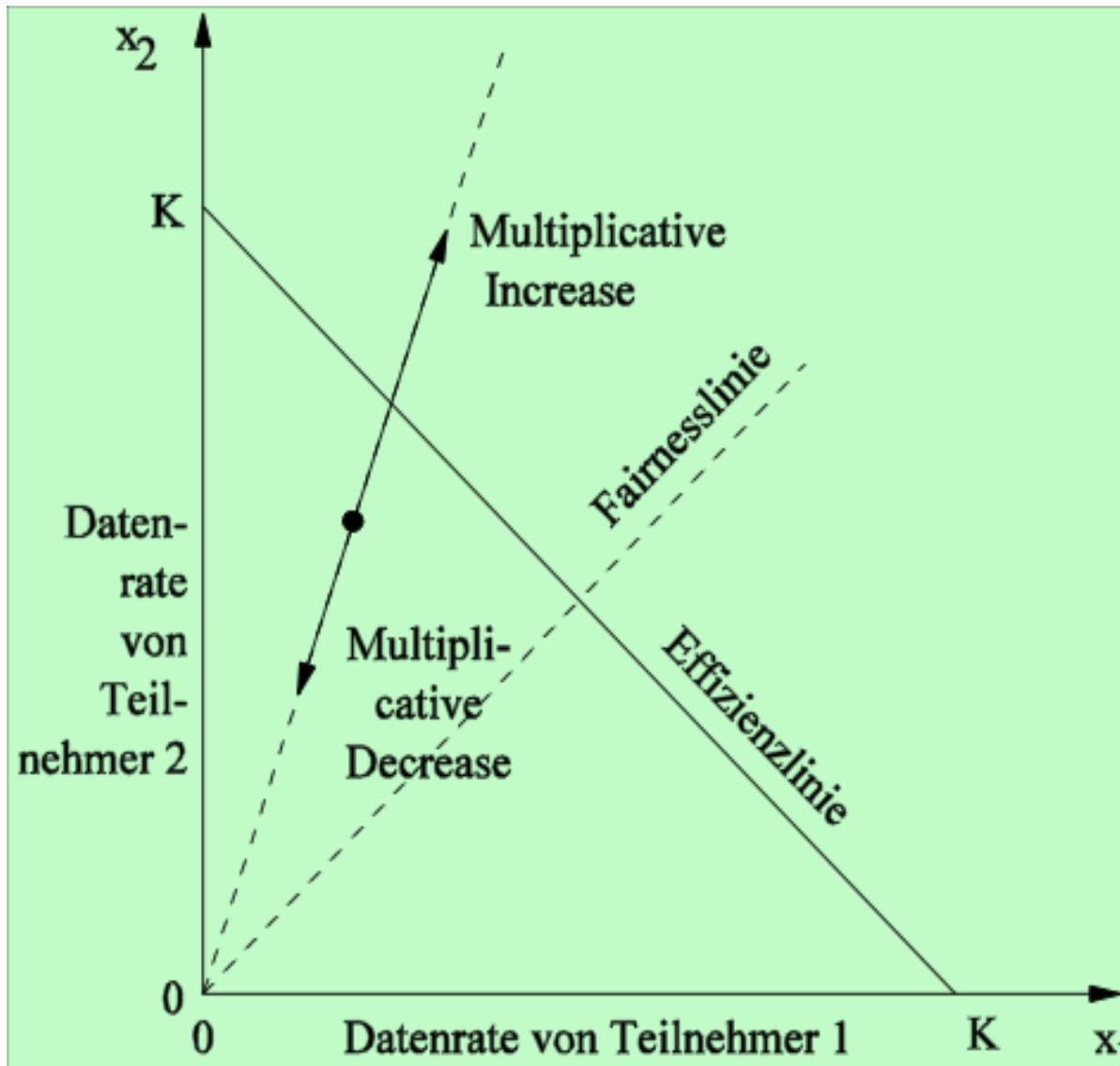
Vektordarstellung (II)



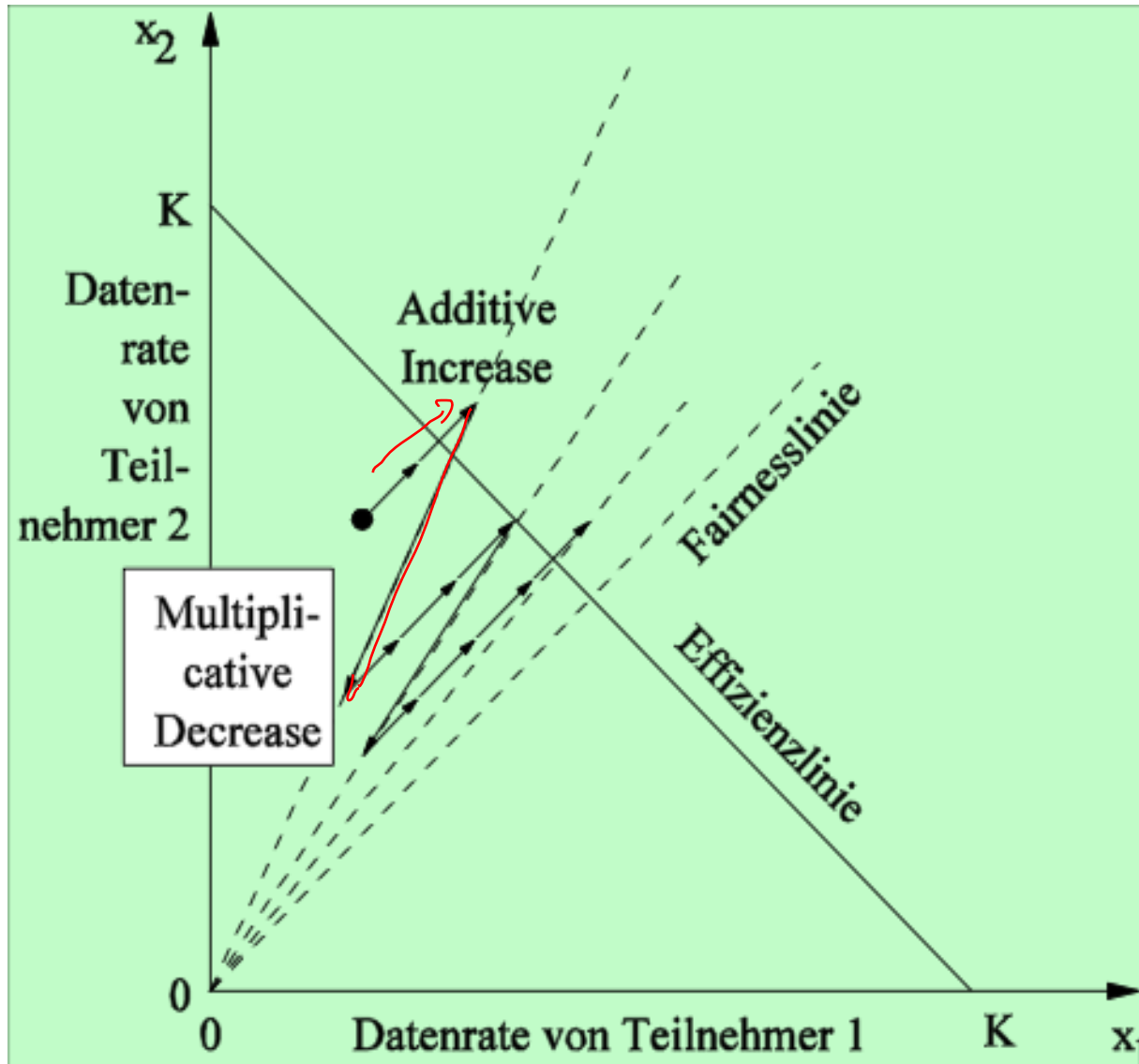
AIAD Additive Increase/ Additive Decrease

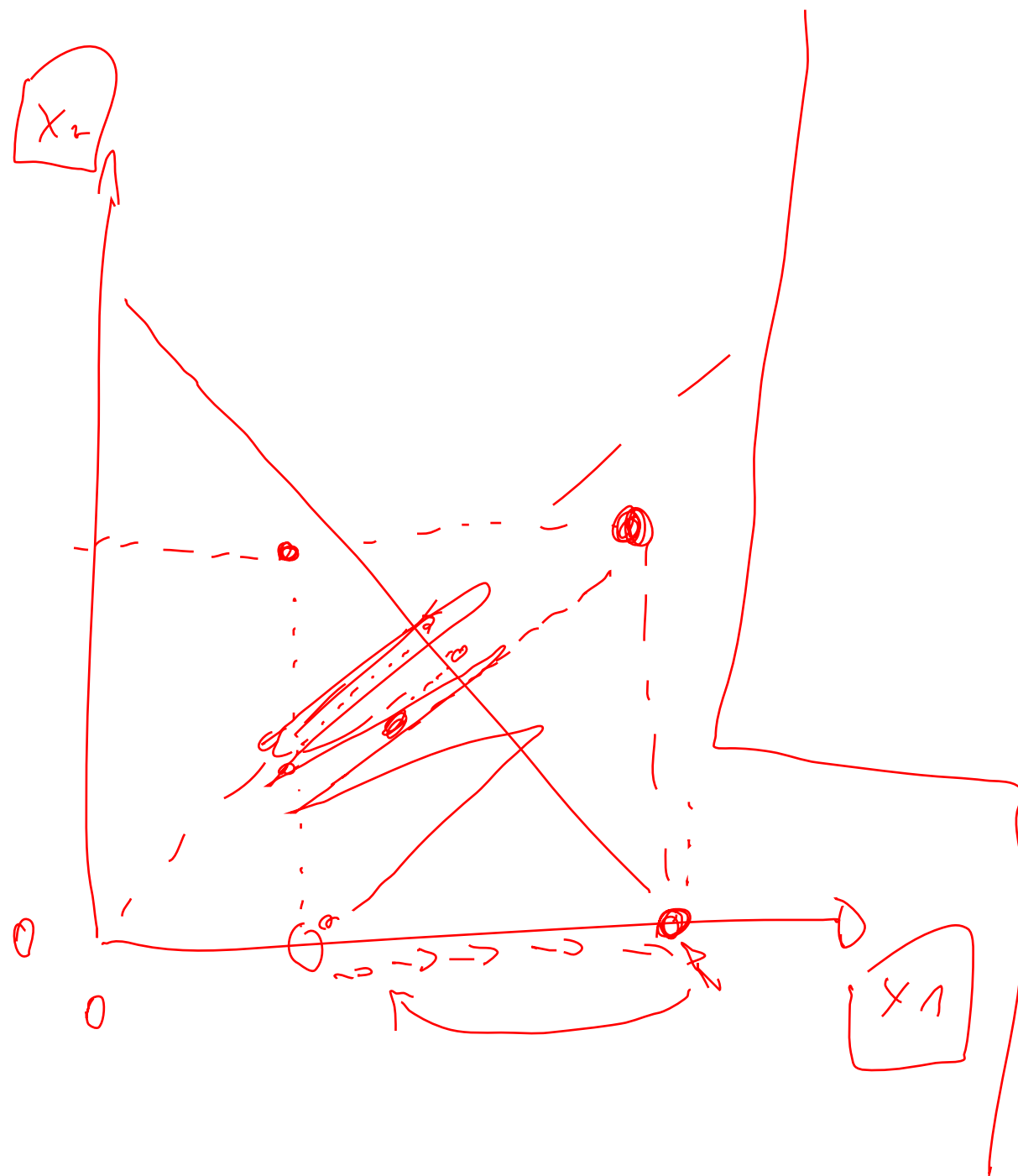


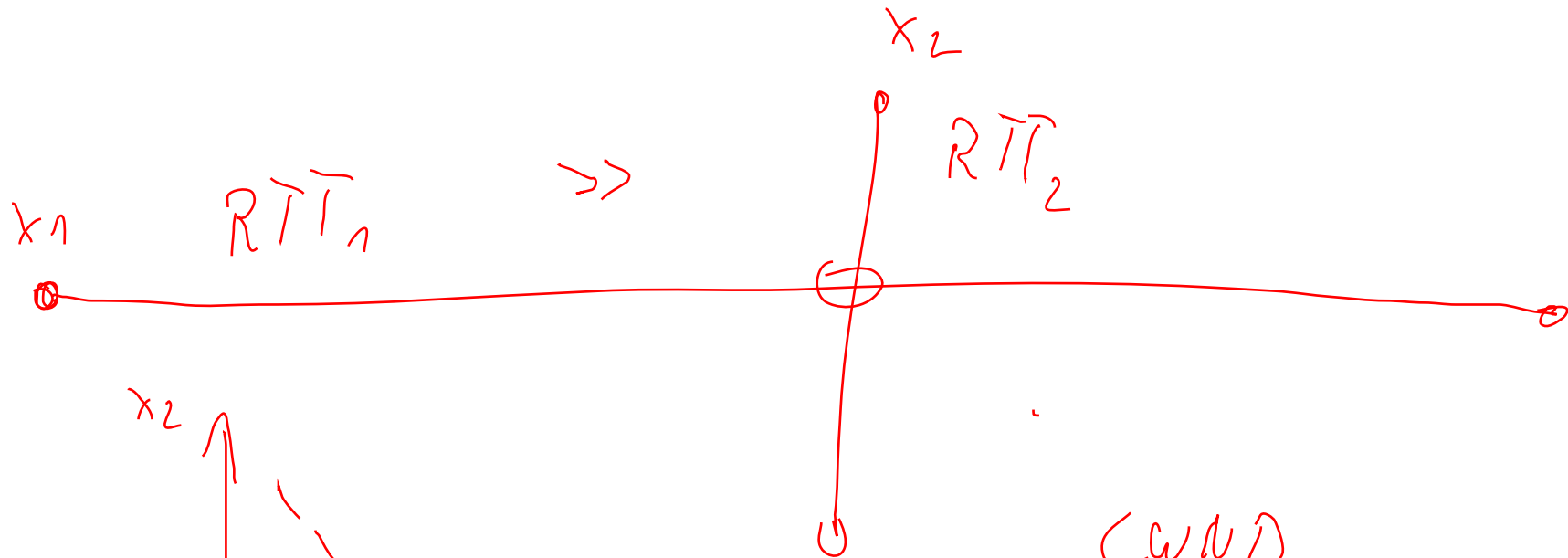
MIMD: Multiplicative Incr./ Multiplicative Decrease



AIMD: Additively Increase/ Multiplicatively Decrease



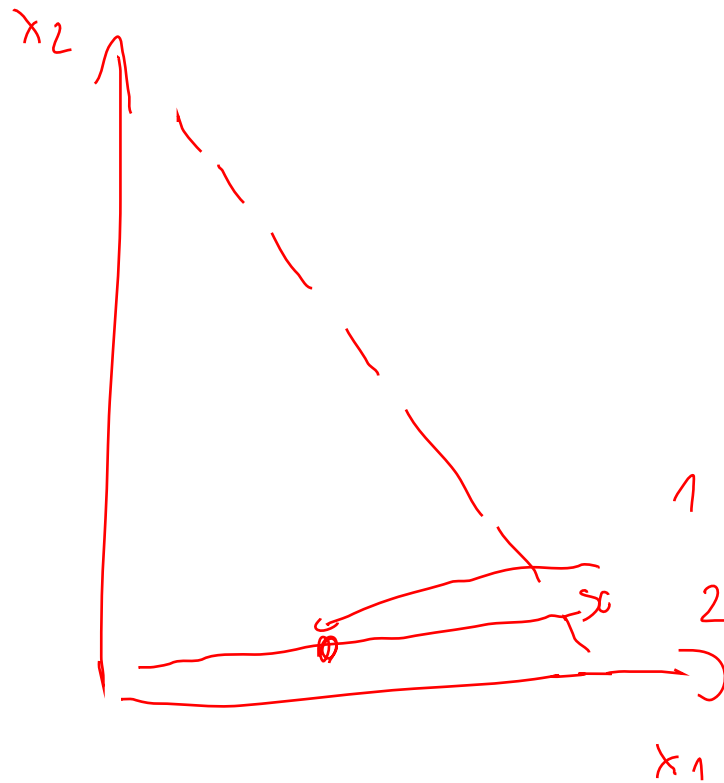




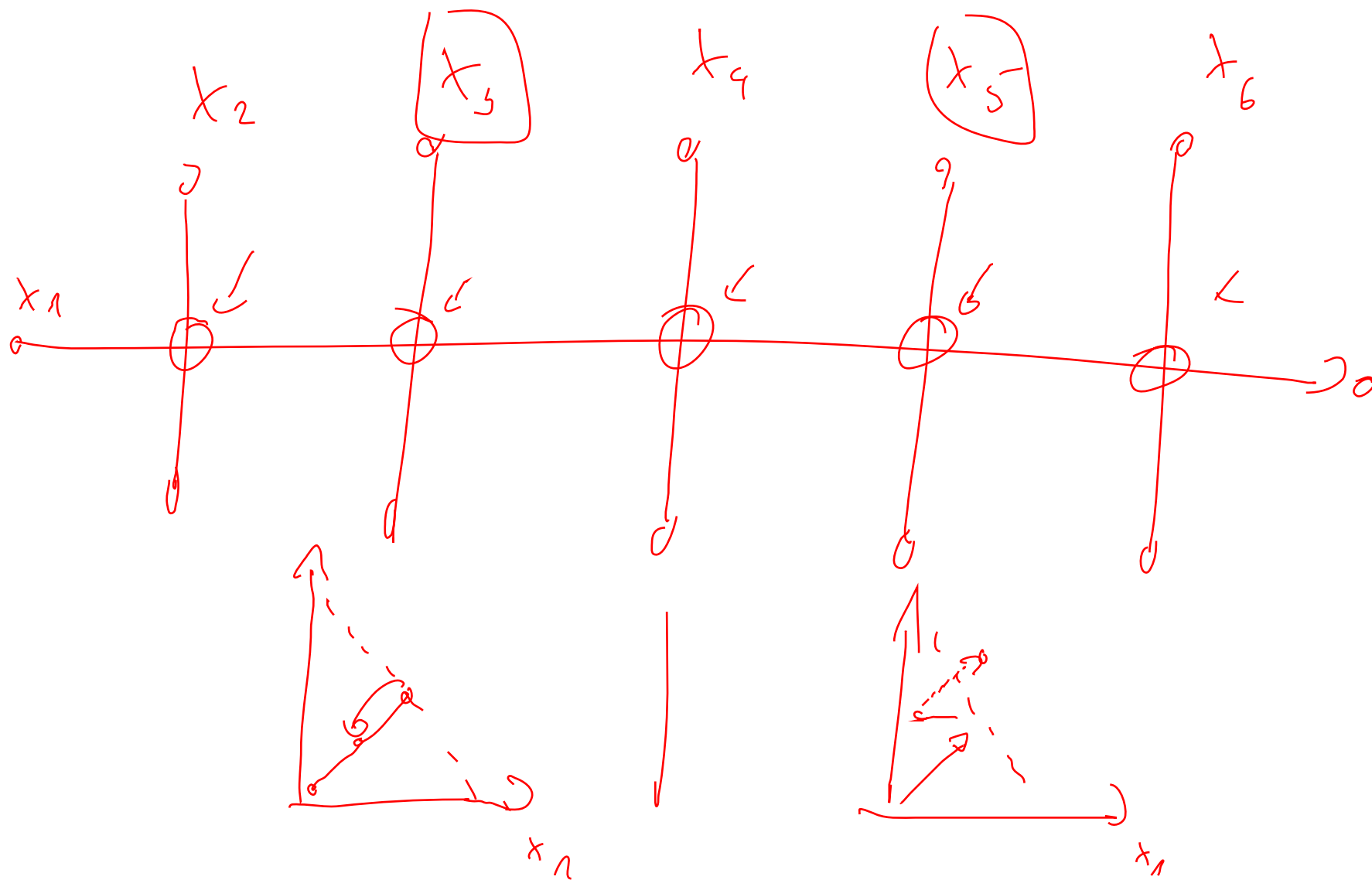
$$\frac{CWND}{RTT} = \text{Datenrate}$$

1 CWND $\xrightarrow{1s}$ 1 2 4 8 16 MSS

2 CWND $\xleftarrow{10s}$ 1 2 4 8 16



1
1
2
1
4



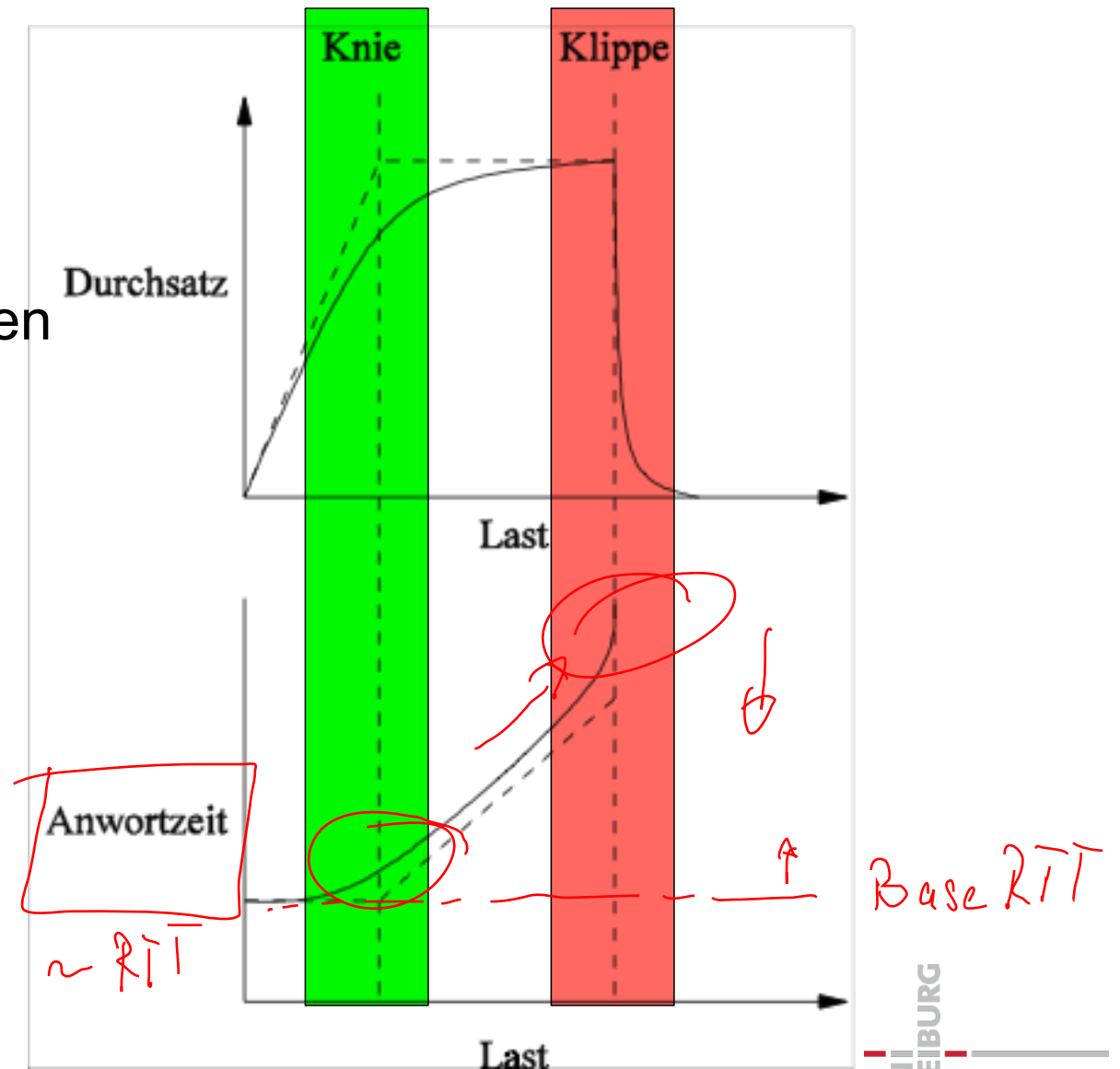
- Verbindungen mit großer RTT werden diskriminiert
- Warum?
 - Auf jeden Router konkurrieren TCP-Verbindungen
 - Paketverluste halbieren Umsatz (MD)
 - Wer viele Router hat, endet mit sehr kleinen Congestion-Window
- Außerdem:
 - Kleinere RTT ist schnellere Update-Zeit
 - Daher steigt die Rate (AI) auf kurzen Verbindungen schneller
 - Mögliche Lösung:
 - konstante Datenratenanpassung statt Fenster-basierte Anpassung

- RTT-basiertes Protokoll als Nachfolger von TCP Reno
 - “L. Brakmo and L. Peterson, “TCP Vegas: End-to-End Congestion Avoidance on a Global Internet”, IEEE Journal on Selected Areas of Communications, vol. 13, no. 8, October 1995, pp. 1465–1480.
- Bessere Effizienz
- Geringere Paketverluste
- Aber:
 - TCP Vegas und TCP Reno gegeneinander unfair



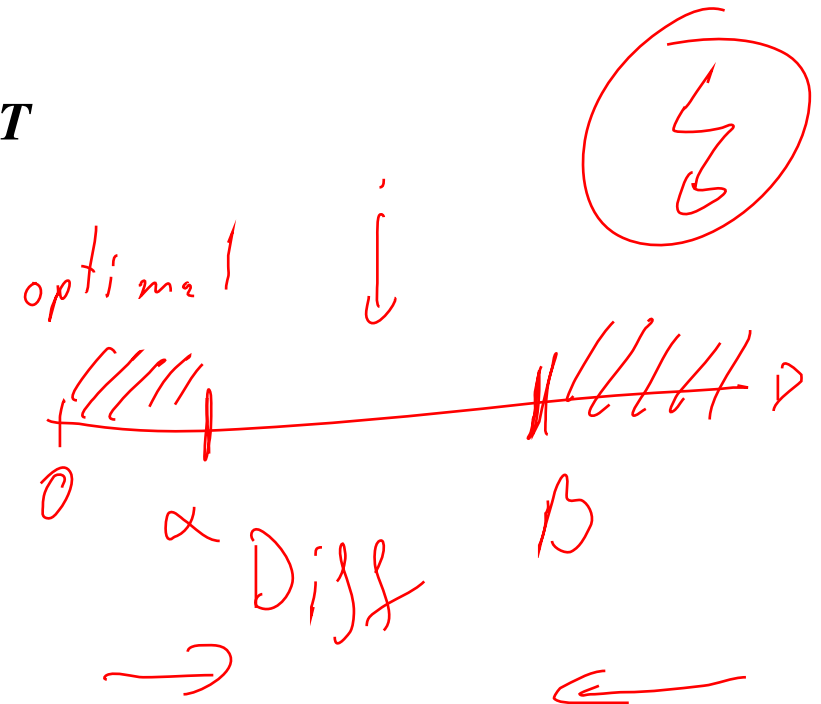
Durchsatz und Antwortzeit

- Klippe:
 - Hohe Last
 - Geringer Durchsatz
 - Praktisch alle Daten gehen verloren
- Knie:
 - Hohe Last
 - Hoher Durchsatz
 - Einzelne Daten gehen verloren



Parameter

- geschätzte Umlaufzeit: RTT
- minimale Umlaufzeit: $BaseRTT = \min \{RTT_1, RTT_2, \dots\}$
- wirkliche Datenrate: $Actual = CWND/RTT$
- erwartete Datenrate: $Expected = CWND/BaseRTT$
- $Diff = (Expected - Actual) BaseRTT \geq 0$
- Programmparameter: $0 \leq \alpha < \beta$
- Wenn $Diff \leq \alpha$ (d.h. $Actual \approx Expected$)
 - Last ist gering
 - $CWND \leftarrow CWND + 1$
- Wenn $Diff > \beta$, (d.h. $Actual \ll Expected$)
 - Last ist zu hoch
 - $CWND \leftarrow CWND - 1$
- Sonst keine Aktion: $CWND \leftarrow CWND$

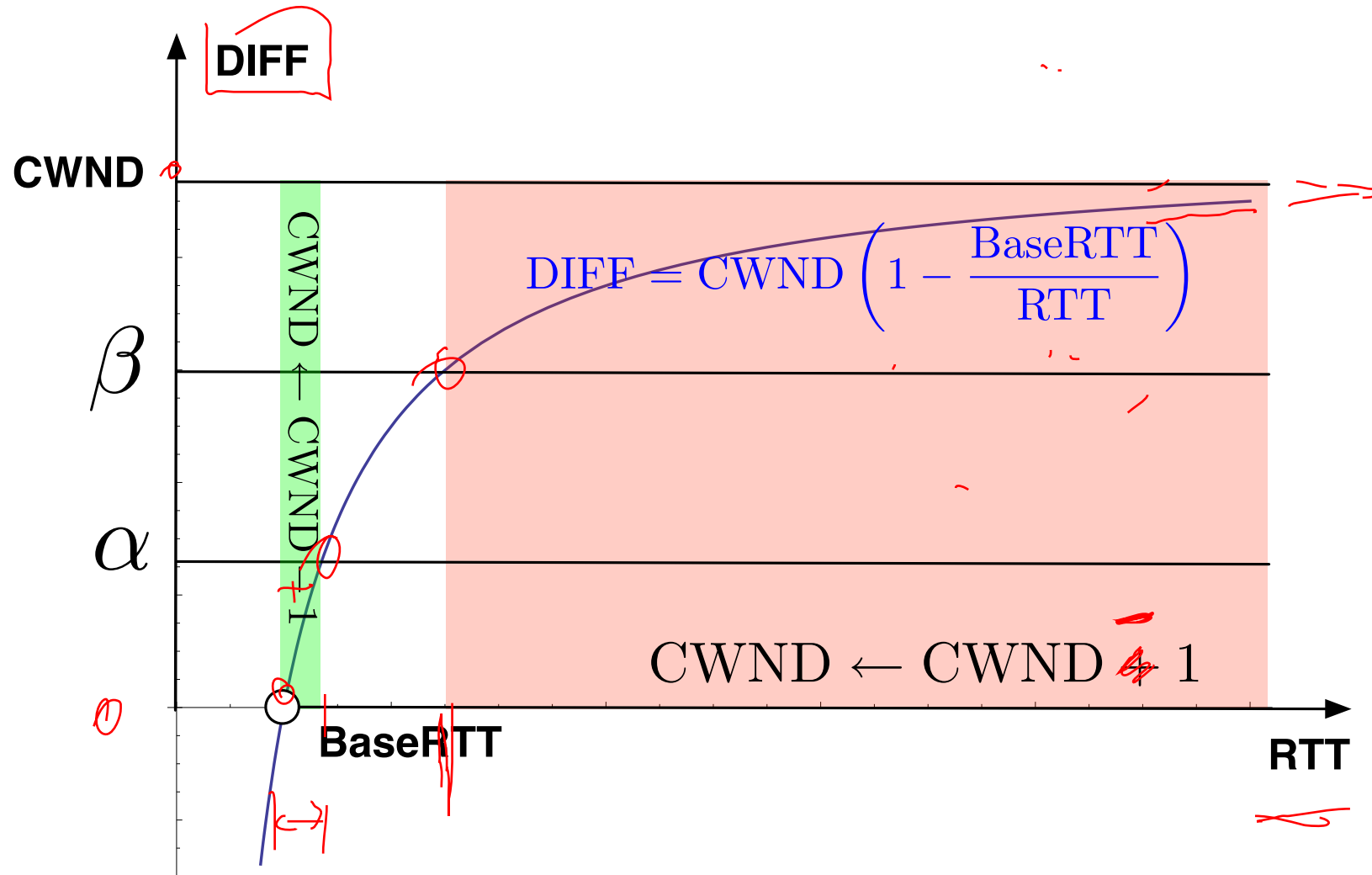


$$D = (Exp - Act) Base RTT$$

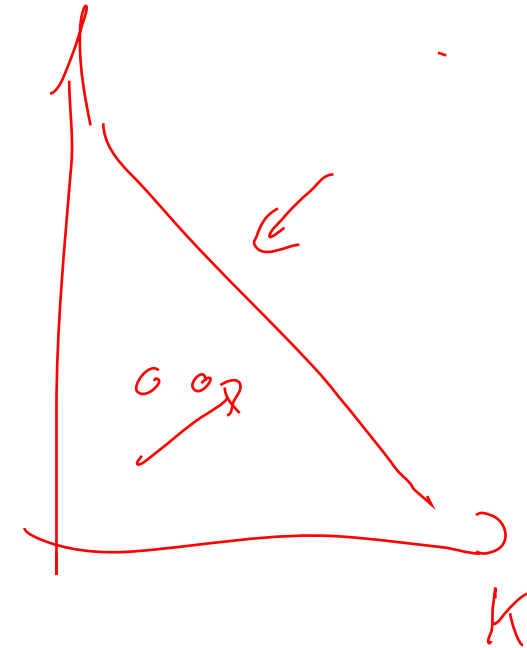
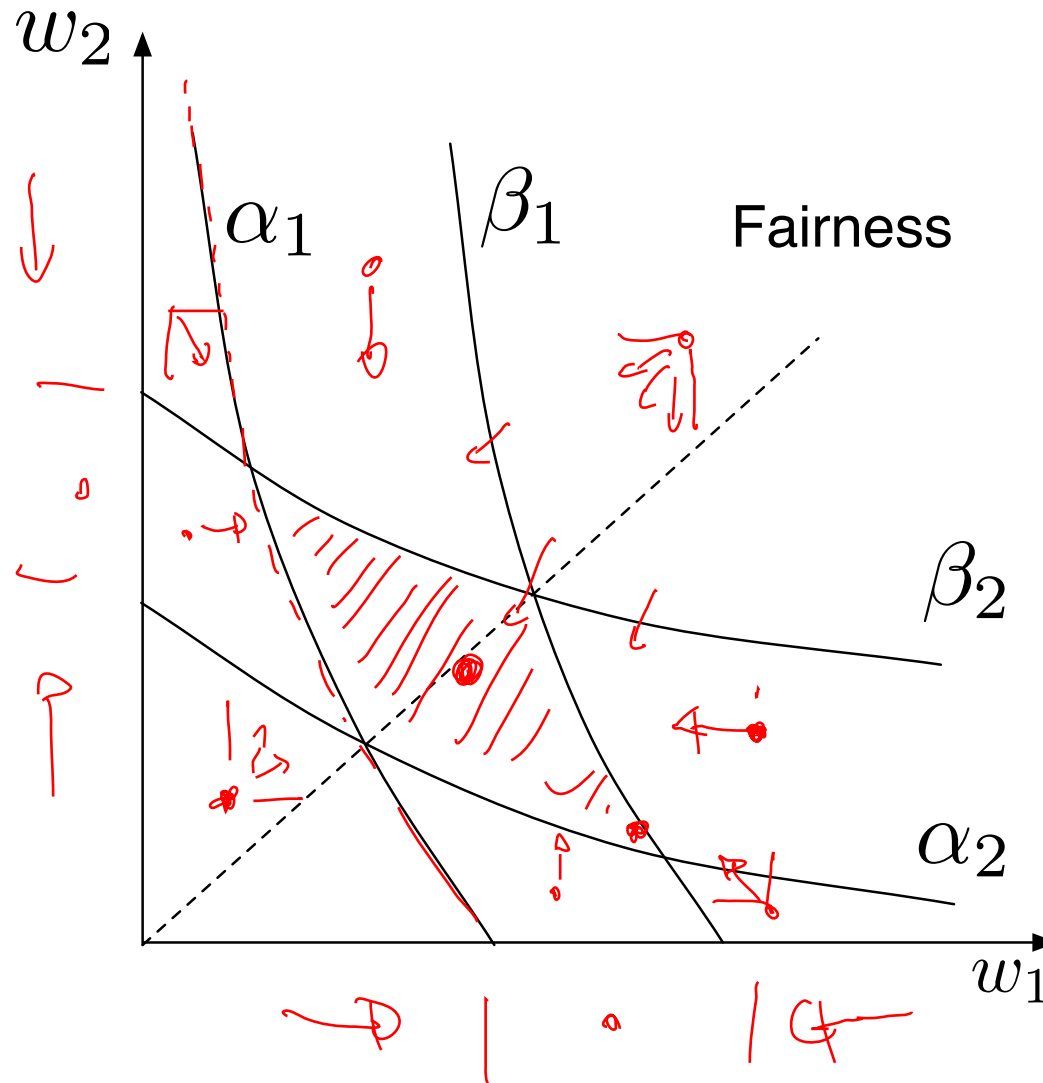
$$= \left(\frac{CWND}{BaseRTT} - \frac{CWND}{RTT} \right) \cdot BaseRTT$$

$$= CWND \left(1 - \frac{BaseRTT}{RTT} \right)$$

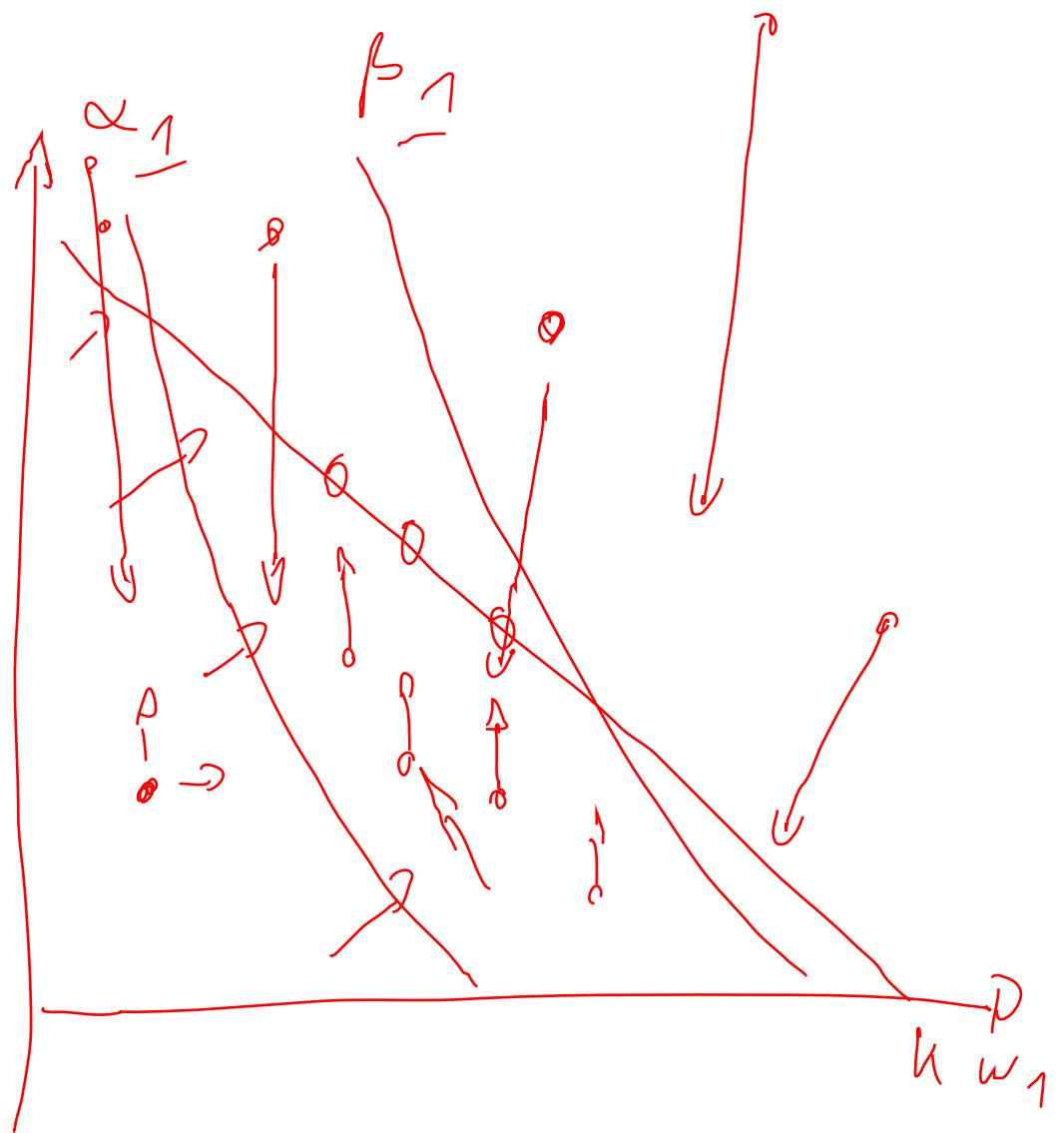
TCP Vegas - Abhängigkeit von RTT



Fenster-Anpassung in Vegas



Reno



Vegas

■ TCP

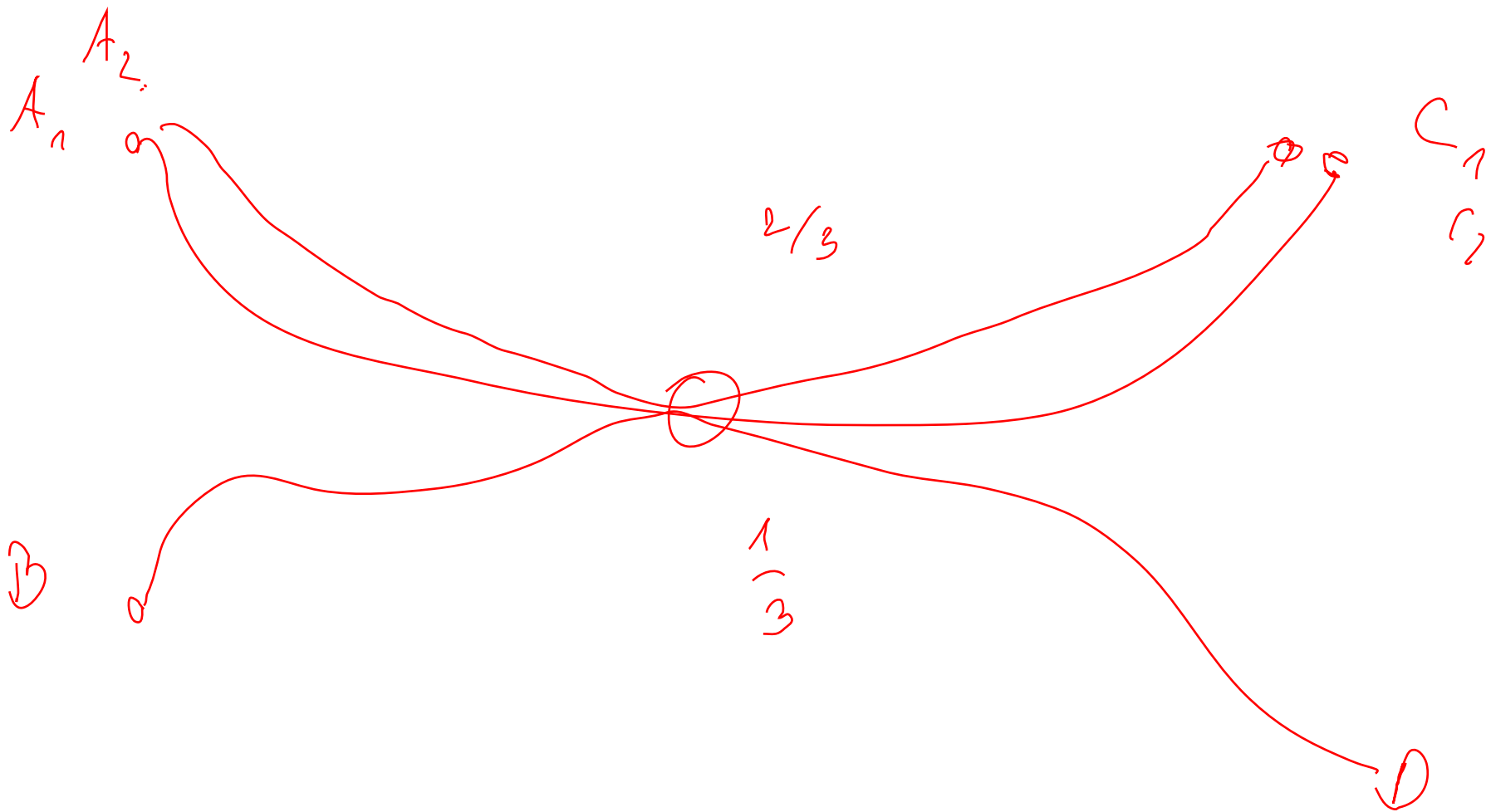
- reagiert dynamisch auf die zur Verfügung stehende Bandbreite
- Faire Aufteilung der Bandbreite
 - Im Idealfall: n TCP-Verbindungen erhalten einen Anteil von $1/n$

■ Zusammenspiel mit anderen Protokollen

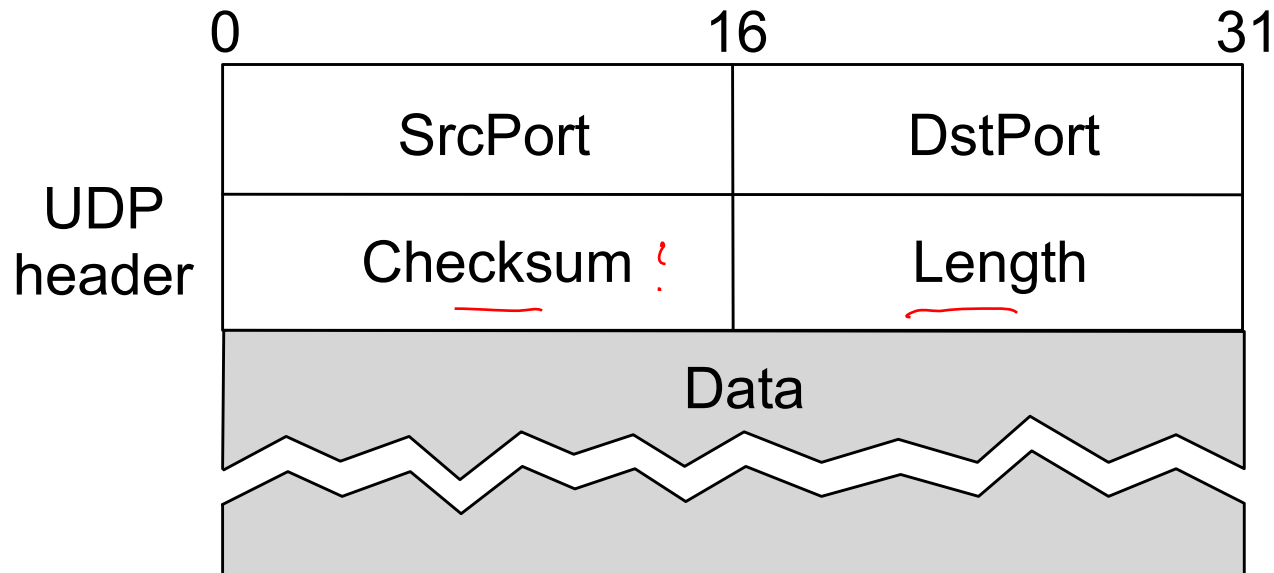
- Reaktion hängt von der Last anderer Transportprotokolle ab
 - z.B. UDP hat keine Congestion Control
- Andere Protokolle können jeder Zeit eingesetzt werden
- UDP und andere Protokoll können TCP Verbindungen unterdrücken

■ Schlussfolgerung

- Transport-Protokolle müssen TCP-kompatibel sein (TCP friendly)



- User Datagram Protocol (UDP) //
 - ist ein unzuverlässiges, verbindungsloses Transportprotokoll für Pakete //
- Hauptfunktion:
 - Demultiplexing von Paketen aus der Vermittlungsschicht
- Zusätzlich (optional):
 - Checksum aus UDP Header + Daten



A
UDP
IPv4

- TCP erzeugt zuverlässigen Byte-Strom
 - Fehlerkontrolle durch “GoBack-N”
- Congestion control
 - Fensterbasiert
 - AIMD, Slow start, *Congestion Threshold*
 - Flusskontrolle durch *Window*
 - Verbindungsaufbau 
 - Algorithmus von Nagle 

Systeme II

5. Die Transportschicht

Christian Schindelhauer

Technische Fakultät

Rechnernetze und Telematik

Albert-Ludwigs-Universität Freiburg